Minimum Physical Requirements for Parallel Execution of the Decimation in Frequency Fast Fourier Transform

Robert J. Meier, Jr.*
Research Scientist

Ames Research Center
National Aeronautics and Space Administration

August 11, 1986

Abstract

When the Fast Fourier Transform of a large number of samples, n, is implemented on physical hardware, performance limits are dictated by fundamental physical laws. These limitations are illustrated by examination of a decimation in frequency Fast Fourier Transform on a cellular layout of n processors. The decimation in frequency algorithm minimizes operation time, $O(\log_2 n)$, but minimizes total execution time only when all other costs, such as communication time are negligible. For a large number of processors ($n > \sim 1800$ with typical current technology), the propagation delay of the decimation in frequency algorithm, $O(n^{1/2} \log_2 n)$, dominates the total execution time. If all costs except propagation delay and operation time are negligible, then total execution time is minimized by choosing $n^{1/2}$ as the radix. Choosing a radix of $n^{1/2}$ creates a two-level nested discrete Fourier transform requiring an execution time of only $O(n^{1/2})$. Other limitations which become significant when the number of samples, n, increases are mentioned. These impose even greater execution times.

Keywords: communication time, fast Fourier transform, shuffling, physical limitations, decimation in frequency, discrete Fourier transform, parallel processing, multiprocessing, propagation delay

* Supported by the National Aeronautics and Space Administration, Ames Research Center

Introduction

As parallel processing developments decrease the cost of processing, other costs such as communication, compilation, and loading become more significant. This paper presents an estimation of the time required to execute a fast Fourier transform[1]. Unlike previous studies[2], consideration will be given to the communication time as well as the operation time. A fast Fourier transform algorithm must be implemented on hardware which exists within the rules of our physical universe[3] and so has finite nonzero size[4]. Furthermore, fundamental physical laws, such as the speed of light, impose limits on their interaction.

This paper first presents a look at the classic decimation in frequency fast Fourier transform, and its operation count. A brief description of an implementation follows. The time to execute the algorithm on the given implementation is evaluated with and without consideration of propagation delay. Because the propagation delay is found to dominate, a description of a modified algorithm using a radix chosen to minimize total execution time is given and shown to be faster. Lastly, the effect of memory size, communication channel capacity, compilation, and alternate implementation is discussed.

Fast Fourier Transform Algorithm

A Fourier transform multiplies a time sample vector, **t**, by a weight matrix, **W**, obtaining a frequency sample vector, **f**, such that

$$\mathbf{f} = \mathbf{W}\,\mathbf{t}, \tag{1a}$$

$$f_k = \sum_{j=0}^{n-1} t_j\, w^{jk}, \tag{1b}$$

$$w = \mathrm{cis}(2\pi/n), \tag{1c}$$

$$\mathrm{cis}(x) = e^{2\pi i x}. \tag{1d}$$

A fast Fourier transform factors the weight matrix, **W**, to dramatically reduce the actual number of operations. Equation (1b) indicates that a discrete Fourier transform requires n^2 additions and n^2 multiplications. The factorization used can be shown with a dataflow graph. Figure 1, a dataflow graph, corresponds to Algorithm 1, the four sample discrete Fourier transform. The line going out the right of each node represents the weighted sum of the data on each line coming from the left. Because all of the weights are powers of w, only the exponent is placed near a line to indicate the weight. Since a weight of one, w^0, occurs most frequently, the absence of an exponent indicates a 0. Note also that because $w^n = 1$, all exponents are modulo n.

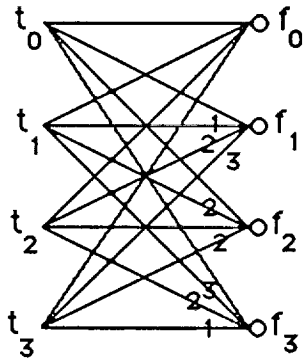

Figure 1
Four Sample Discrete Fourier Transform

$$f_0 = t_0 + t_1 + t_2 + t_3$$
$$f_1 = t_0 + w^1 t_1 + w^2 t_2 + w^3 t_3$$
$$f_2 = t_0 + w^2 t_1 + t_2 + w^2 t_3$$
$$f_3 = t_0 + w^3 t_1 + w^2 t_2 + w^1 t_3$$

Algorithm 1
Four Sample Discrete Fourier Transform

Figure 2 and Algorithm 2, illustrate how the matrix is factored for the fast Fourier transform. Along any path from a time sample to a frequency sample, the sum of the exponents, corresponding to the product of the weights, is the same as for the discrete Fourier transform above. Note that while the discrete transform requires twelve nontrivial additions and eight nontrivial multiplications, the fast transform requires only eight nontrivial additions and five nontrivial multiplications.

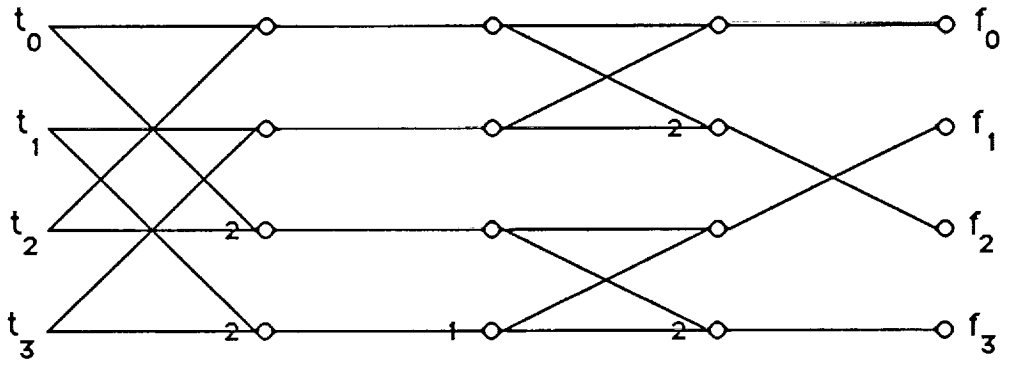

Figure 2
Four-Sample Fast Fourier Transform

$$g_0 = t_0 + t_2$$
$$g_1 = t_1 + t_3$$
$$g_2 = t_0 + w^2 t_2$$
$$g_3 = t_1 + w^2 t_3$$

$$g_0' = g_0$$
$$g_1' = g_1$$
$$g_2' = w^0 g_2$$
$$g_3' = w^1 g_3$$

$$g_0'' = g_0' + g_1'$$
$$g_1'' = g_0' + w^2 g_1'$$
$$g_2'' = g_2' + g_3'$$
$$g_3'' = g_2' + w^2 g_3'$$

$$f_0 = g_0''$$
$$f_1 = g_2''$$
$$f_2 = g_1''$$
$$f_3 = g_3''$$

Algorithm 2
Four Sample Fast Fourier Transform

The decimation in frequency algorithm can be extended indefinitely, characterized by two parameters as shown in Figure 3. Given the number of samples, n, and the number of terms in each weighted sum, the radix, r, the algorithm is completely determined. All but the last stage has a butterfly substage and a phase substage. The last stage has a butterfly, and a decimator. Each butterfly performs a weighted sum. Each multiplier in the phase substage multiplies its single input line by a power of w. The decimator substage rearranges the samples.

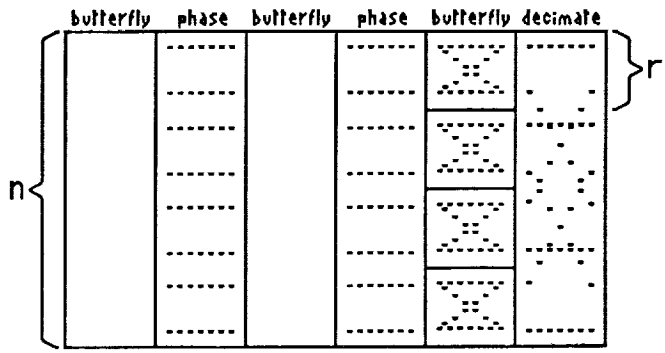

Figure 3
Parameters of Fast Fourier Transform

The general algorithm, Algorithm 3, consists of $\log_r n$ stages. During stage i, it uses internal vectors, $\mathbf{g}^{(2i-1)}$, and $\mathbf{g}^{(2i)}$. It also uses internal indices, a, b, c, and d, as well as vector, **p**. All but the for* loop can be executed in parallel.

for $0 \le k < n$,

$g^{(0)}_k = t_k$

for* $0 < l < \log_r n$,

$a^{(l)} = n/r^{l-1}$,

$b^{(l)} = n/r^l$,

$c = n/r$

$d^{(l)} = r^{l-1}$

for $0 \le i < n/a^{(l)}$,

for $0 \le j < b^{(l)}$,

for $0 \le k < r$,

$$g^{(2l-1)}_{ai+j+bk} = \sum_{m=0}^{r-1} w^{ckcm}\, g^{(2l-2)}_{ai+j+bm}$$

$$g^{(2l)}_{ai+j+bk} = w^{djck}\, g^{(2l-1)}_{ai+j+bk}$$

$c = n/r$,

$l = \log_r n$,

for $0 \le i < c$,

for $0 \le k < r$,

$$g^{(2l-1)}_{ci+k} = \sum_{m=0}^{r-1} w^{ckcm}\, g^{(2l-2)}_{ci+m}$$

$l = \log_r n$,

for $0 \le k < n$,

$$f_k = g^{(2l-1)}_j \qquad \text{where } j = \sum_{i=0}^{l-1} p_i\, r^i$$

$$\text{and} \qquad k = \sum_{i=0}^{l-1} p_i\, r^{l-i}$$

Algorithm 3
Decimation in Frequency Fast Fourier Transform

Operation Time

The simplest figure to derive from the given algorithm is the number of additions and multiplications that must be performed. The only nontrivial additions occur during summation and number $n \log_r n\,(r-1)$. The nontrivial multiplications during summation number $n \log_r n\, k/r$ where k is the number of nonzero elements in the modulo r multiplication table. If r is prime, then $k = (r-1)^2$. If r is a power of 2, 2^e, then $k = 4^e(1-2^{-e}) - e2^{e-1}$. The number of nontrivial multiplications in the phase substages is $n\,(\log_r n - 1)\, k/r^2$.

We can examine the time involved in each stage. In the following discussion we consider only the time to add, T_a, and the time to perform a multiplication, T_m. The total weighted sum multiplication time, T_{M1}, the total addition time, T_A, and the total adjusted multiplication time, T_{M2}, can be calculated. If there are enough cells, then all the weighted sum multiplications can be performed in parallel as shown in Figure 5. If not, then the minimum time is achieved by distributing the work evenly. (N.B. cl(x) is the smallest integer greater than or equal to x.)

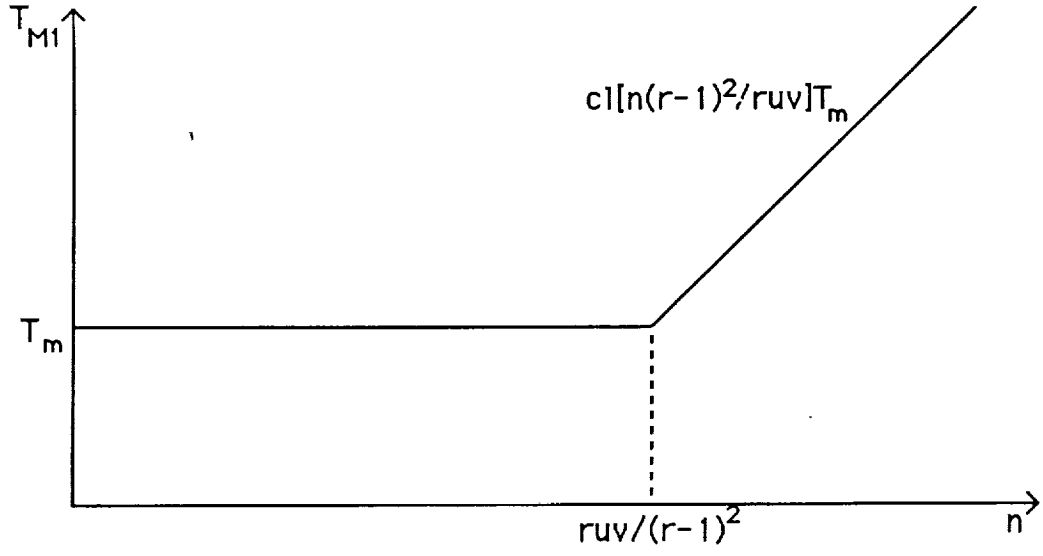

Figure 5
Weighted Sum Multiplication Time versus Number of Samples

If there are enough cells, the additions can be performed in binary tree fashion as shown in Figure 6. Figure 7 shows how the addition time increases with the number of samples. The steps in Figure 7 reflect the depth of the addition tree that can be supported for the given number of samples.

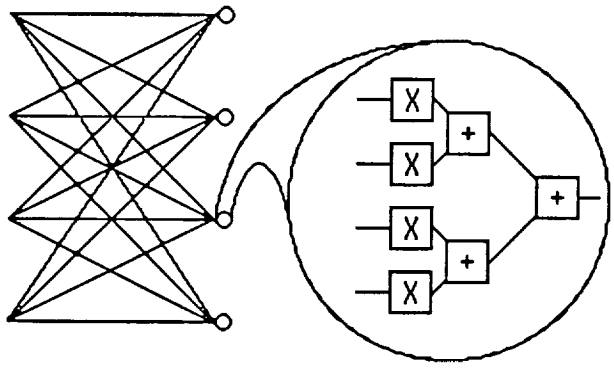

Figure 6
Detail of Binary Tree Addition at Butterfly Node

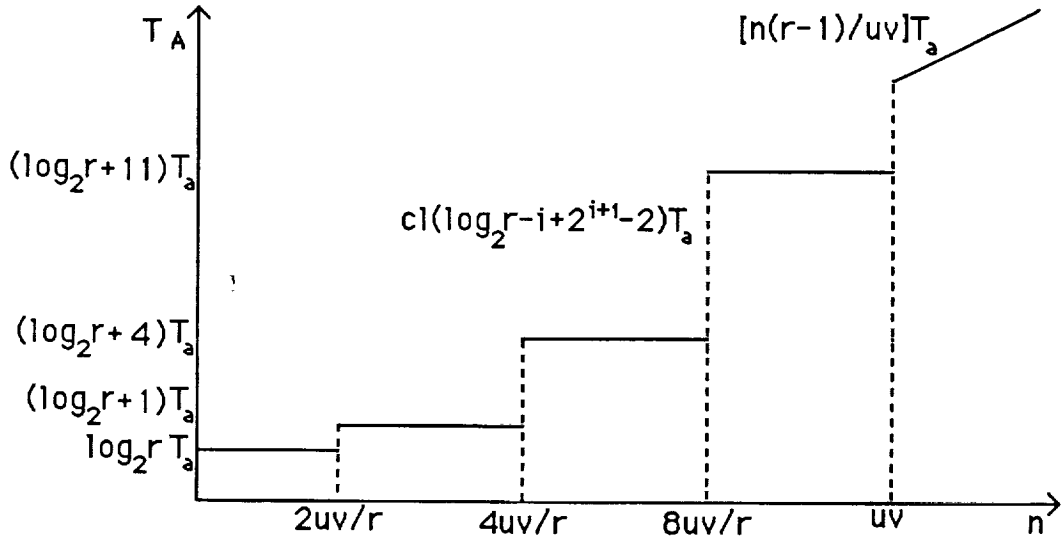

Figure 7
Addition Time versus Number of Samples (r=8)

The phase multiplications are independent and can be done in parallel in time, T_m, if there are enough cells. This is shown in Figure 8.

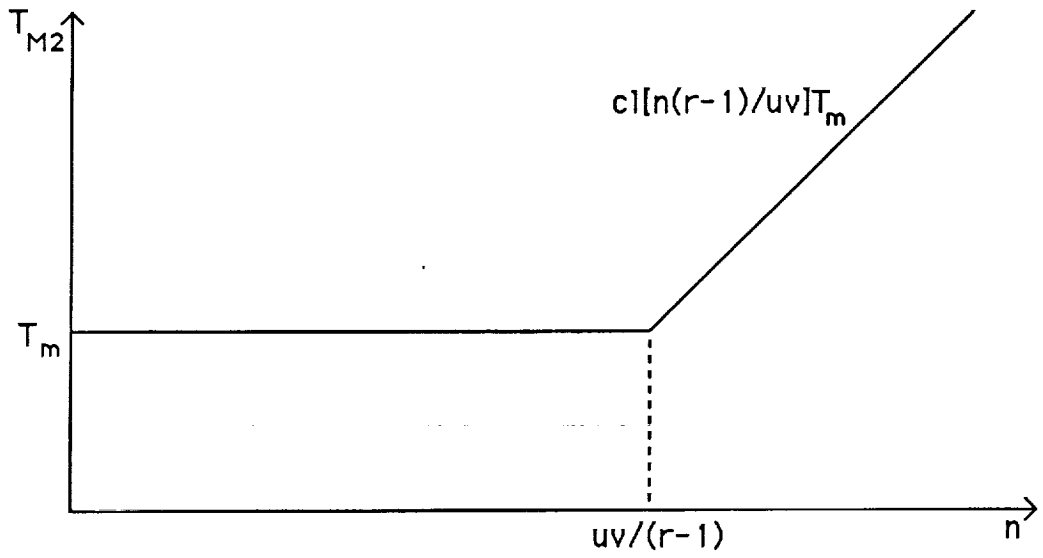

Figure 8
Multiplication Time versus Number of Samples

The total time for all stages is the sum of the time for each stage.

$$T = T_{M1}\log_r n + T_A\log_r n + T_{M2}(\log_r n - 1) \quad (2)$$

If we assume that we always have enough cells, $uv \geq n(r-1)$, then the minimum total time is achieved,

$$T_{min} = cl(\log_r n)\,[2T_m + cl(\log_2 r)T_a] - T_m \quad (3)$$

Implementation

For the purposes of this discussion, a simple two-dimensional layout with a fixed control structure will be assumed. Such an implementation could be laid out in VLSI and will serve to illustrate the points involved. The entire system will be assumed to be made up of simple cells that contain a complex multiplier, a complex adder, a memory store, and a communications interface as shown in Figure 4. Though as many as three dimensions have been used for layout, normal fabrication replicates devices primarily in two dimensions. It is common to have thousands of devices juxtaposed in two dimensions and only a few in the third. Consideration of unrestrained replication in three dimensions changes the results slightly quantitatively, but not qualitatively.

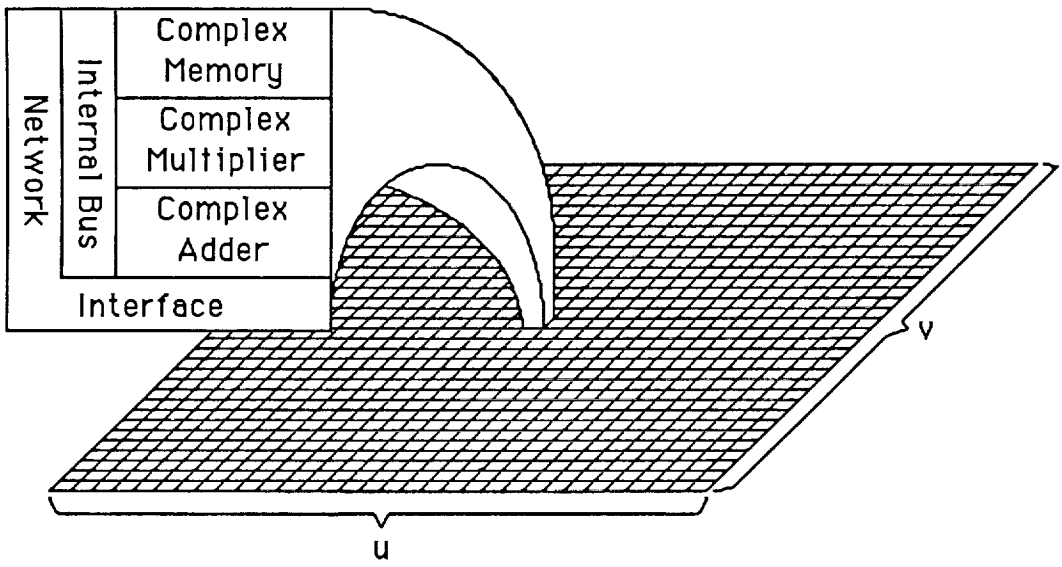

Figure 4
Cellular Layout of Fast Fourier Transformer

For the purposes of this discussion, the array will be assumed to be rectangular with u by v cells. For simplicity, u and v are powers of r. Note that the network interfaces form a continuous mesh over the entire structure. It is assumed that only one element of each cell can operate at any given time, though all of the cells can operate simultaneously. The choice of a different network will change the quantitative results slightly, but not the qualitative results. Even when a logical hypercube with d dimensions is used, it must still be implicitly mapped onto a physical network of two dimensions.

For the purposes of this discussion, interprocessor communication time is assumed to be one system clock cycle. This system clock cycle is assumed to be equal to the maximum signal propagation time, p_{max}. To guarantee synchronization, such a clock cycle must be at least equal to $p_{max}-p_{min}+T_s$. (p_{max} and p_{min} are the maximum and minimum signal propagation times respectively. T_s is a setup, switching, or settling time independent of the size of the structure.) As p_{max} grows $p_{max}-p_{min}+T_s$ must grow. The difference, $p_{max}-p_{min}$, is the sum of individual differences along a chain. At each stage along the chain, the difference is >0 and so p_{min} cannot keep up with p_{max} and the difference, $p_{max}-p_{min}$ must eventually be dominated by p_{max}. Multilevel clock and wait state schemes abound but these cannot decrease the communication times below propagation delay. This inherent dependence on the maximum propagation delay, implies that choice of clock scheme can determine the low-order terms and the leading constant in the formulae, but cannot affect the order of the leading term. The crossover points encountered in comparing different algorithms might change, but the asymptotic behavior is unchanged.

Propagation Time

In this section we will consider communication time. In the previous section, we implicitly assumed that all the data was available, when and where it was needed. This was equivalent to the assumption that communication time was identically zero. As we make our processors faster, either by improved technology or by locally parallel operations, the cost of communication becomes relatively more significant.

Communication time is strongly dependent on the physical dimensions of the device, and so we will now examine this aspect. We stated that the structure is u x v cells but we neglected to consider the size of the cells themselves. For the sake of this discussion, let us assume, that each cell is square and has a side of length a as shown in Figure 9.

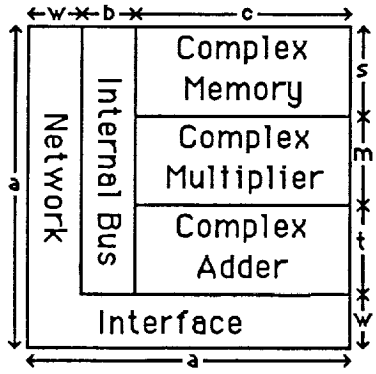

Figure 9
Physical Dimensions of Cell

In general signal propagation speed is limited absolutely by the speed of light, practically by a technology dependent constant that we will call c. Almost all machines built support equal access time to all parts of the machine. They enforce it by simply imposing a minimum time on all transmissions equal to the worst case transmission time. In the model given, the worst case, assuming no contention, is that of corner to corner communication,

$$T_c = (u+v)a/c. \qquad (4)$$

For a given number of cells, uv, this is minimized if $u = v$.

If we examine Algorithm 3, we can count the number of necessary communications, in like manner to our count of the number of operations. With enough processors, $u^2 \geq n(r-1)^2/r$, the weighted sum multiplications can be performed independently, and we assume that fetching the data will be done independently in $T_c = 2(r-1)an^{1/2} / cr^{1/2}$. If we assume a binary tree summation as shown in Figure 10, only $\log_2 r$ global transmission times are necessary.

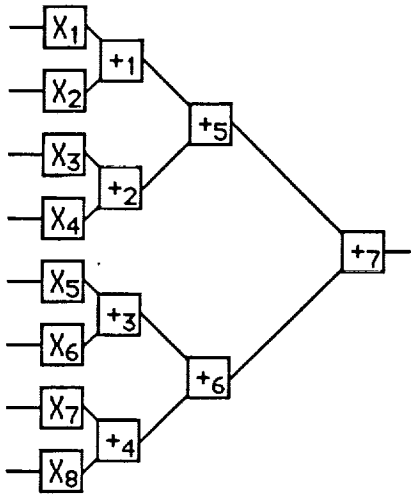

Figure 10
Dataflow Diagram of Radix 8 Weighted Sum

Figures 8a, 8b, 8c, and 8d show how this radix 8 weighted sum would be mapped onto a set of processors in our array. The arrows indicate the intercell communication that must take place. By judicious mapping of the operations from the dataflow graph to the processors, some of the data can be in the right cell without the need for a global transmission.

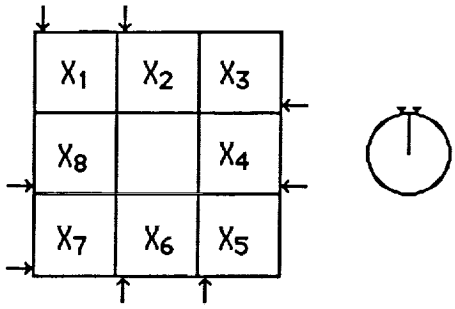

Figure 11a
Weighted Sum Multiplications in First Time Interval

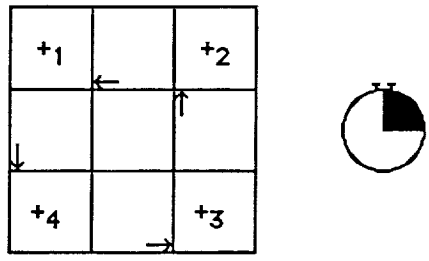

Figure 11b
Weighted Sum Additions in Second Time Interval

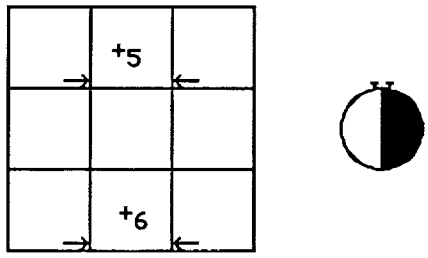

Figure 11c
Weighted Sum Additions in Third Time Interval

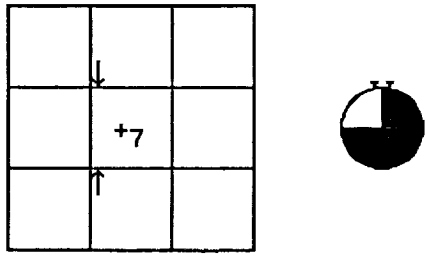

Figure 11d
Weighted Sum Addition in Fourth Time Interval

With the above processor mapping in mind, the total execution time, including propagation time is,

$$T_{min} = cl(\log_r n)\,[2T_m + cl(\log_2 r)T_a + cl(\log_2 r)T_c] - T_m, \quad (5)$$

and since T_c is a function of n,

$$T_{min} = n^{1/2}\, cl(\log_r n)\, cl(\log_2 r)\, 2(r-1)a \,/\, cr^{1/2} + cl(\log_r n)\,[2T_m + cl(\log_2 r)T_a] - T_m \quad (6)$$

Figure 12 illustrates the combination of operation time and propagation time. The total time is dominated by operation time for small n and is dominated by propagation time for large n. For typical technology[5], ($T_m \approx T_a \approx 80ns$, $c \approx 20Mm/s$, $a \approx 8cm$, $r = 2$) the crossover occurs when $n \approx 1{,}800$.

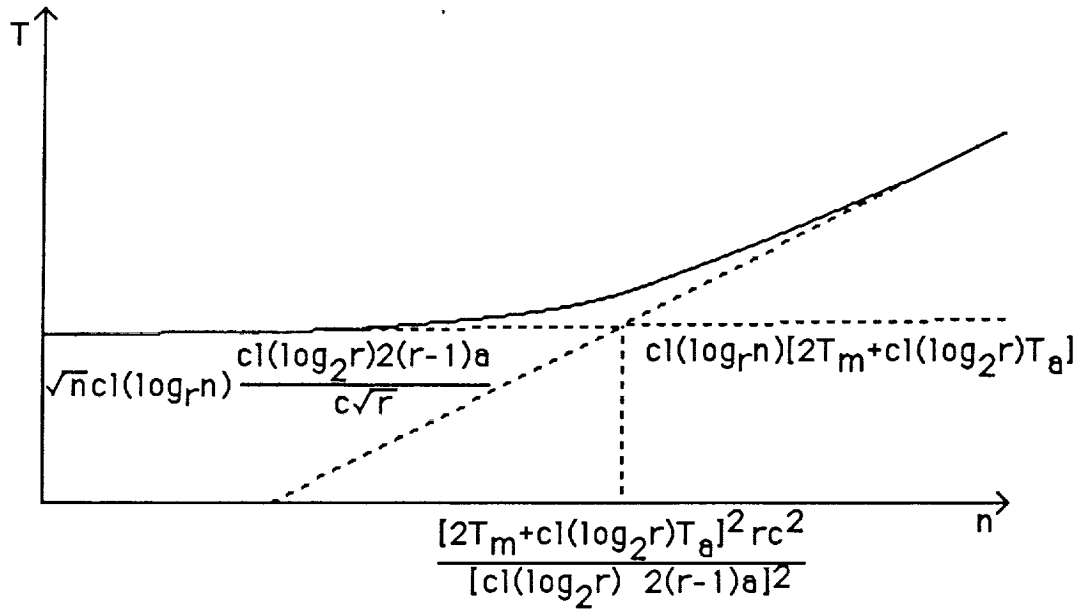

Figure 12
Total FFT Execution Time versus Number of Samples

Choice of Radix

In this section we consider the choice of radix, r, that minimizes total execution time. In the previous section, we implicitly assumed that the radix was a constant independent of n. Normally, the radix has been chosen to be exactly two, the choice which leaves no processors idle at any time. As we increase the radix, we support more local storage of intermediate results and thus expect to reduce the communication load. We will thus consider the effect of changing radix for small numbers of samples where processing time dominates and for large numbers of samples where communication time dominates.

Figure 13 shows the effect of changing the radix for fixed n as indicated by the formula in Figure 9. For small n, the effects of truncation are pronounced and a radix exists which minimizes total time. If multiplication time and addition time are equal, the optimum radix is approximately 2 for reasonable n ($16 < n$). With typical technology, the multiplication time is about thrice addition time and the optimum radix is approximately 4.

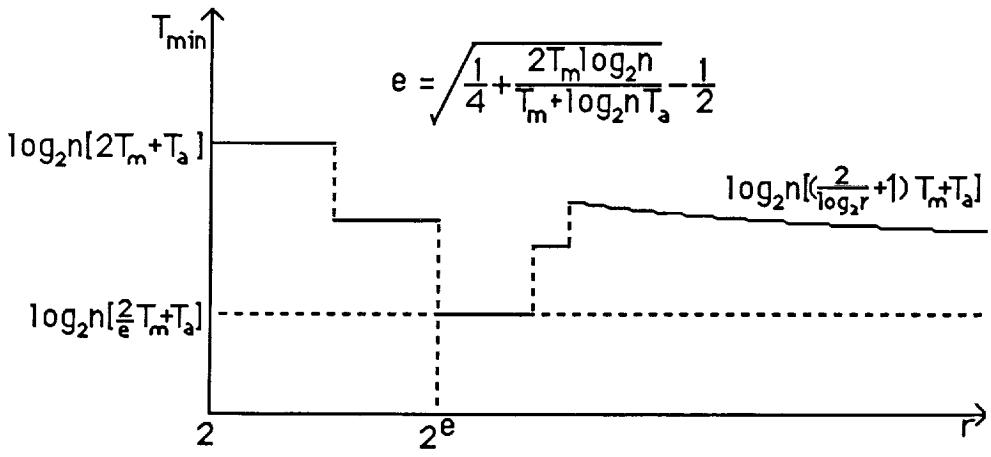

Figure 13
Total Execution Time versus Radix for Small Sample Number

For large n, where the communication time dominates, the weighted sum additions need not be done in a binary tree. If one processor for each butterfly is used to perform the additions sequentially , the addition and multiplication time at each stage is respectively $(r-1)T_a$ and rT_m. With this is mind, T_c and T_{min} can be recalculated.

$$T_{min} = cl(\log_r n)\,[rT_m + (r-1)T_a + T_c] - T_m \qquad (7a)$$

$$T_c = n^{1/2}\, a/c \qquad (7b)$$

Figure 14 shows how the minimum execution time for large n varies with radix under this assumption. Note that the minimum occurs when the radix equals $n^{1/2}$ and hence there are only two stages as shown in Figure 15. If the radix is larger than $n^{1/2}$, the time to perform the additions sequentially exceeds the communication time. If the radix is smaller than $n^{1/2}$, the communication time exceeds the sequential addition time. Note also that the minimum assuming two stages is less than the minimum shown in Figure 9 for the ordinary decimation in frequency algorithm.

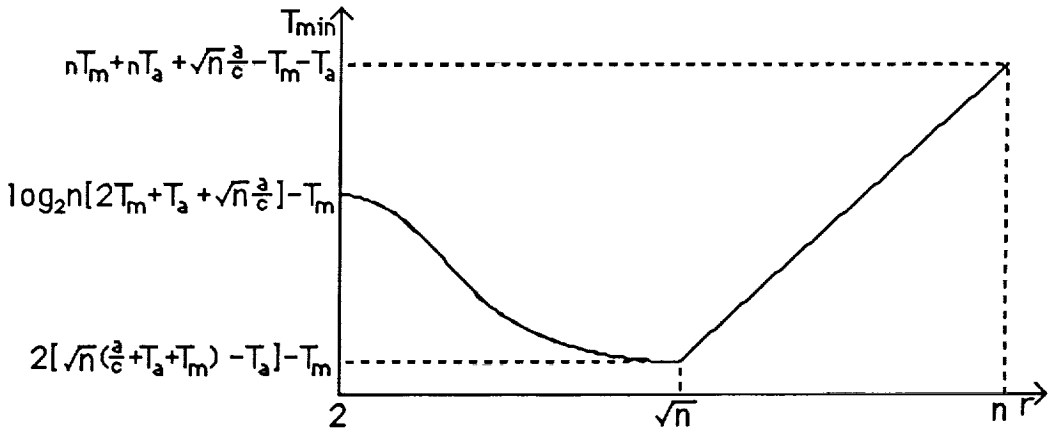

Figure 14
Execution Time versus Radix for Large Sample Number

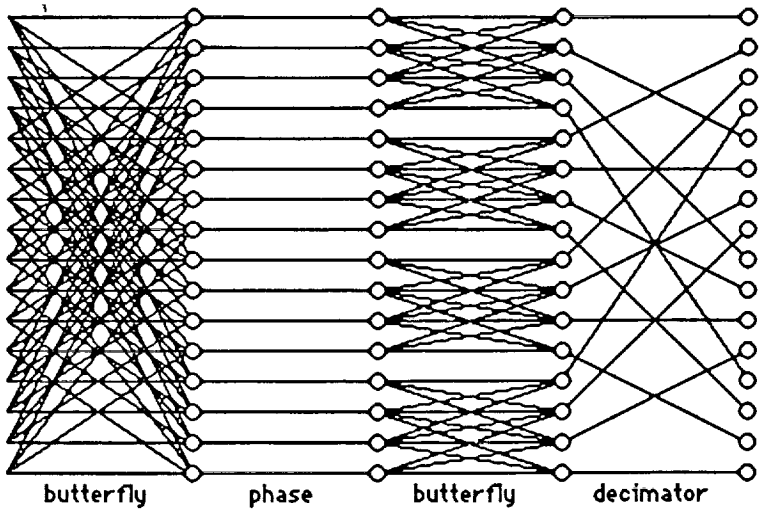

Figure 15
Two-Stage 16 Sample Fourier Transform

Other Limits

The evaluation of the two-stage nested discrete Fourier transform in the last section implicitly assumed that the memory available at each node was infinite (or that memory occupied infinitesimal space). In fact, as the radix is increased to $O(n^{1/2})$, the memory in each cell must increase to $O(n^{1/2})$ and so must the cell size to $O(n^{1/4})$. This drives the propagation delay to $O(n^{3/4})$. Even if there is no need to store $n^{1/2}$ intermediate results, it is necessary to store the weights. The decimation in frequency algorithm does not escape this fate, since it must store weights as well and the number grows with the number of stages.

In addition to the growth of the memory section of each cell, an increase in radix to $O(n^{1/2})$ drives the number of signals in the system from $O(n)$ to $O(n^{3/2})$. If the number of nodes remains at n, each network element must handle a load increasing as $O(n^{1/2})$. This does not become a dominant problem however, because the execution time is growing as $O(n^{1/2})$ as well.

Compilation must be considered as well. If all of the weights are precomputed then there are n log n such constants or n log n pointers to n such constants. For a large enough n, the storage of n log n constants would require a structure with a radius, propagation delay, and execution time of $O((n \log n)^{1/2})$. If these constants are computed during execution, then $O(\log n)$ operations at each node are required to compute them (or their pointers). The tradeoff between simplifying the decimation stage, reducing the precomputation of constants, and reducing the storage of constants is complicated, but eventually it must impose a limit between $O(n^{1/2})$ and $O(n^{3/4})$.

Conclusion

In this paper we have presented a representative fast Fourier transform algorithm and implementation to illustrate that as the number of samples, n, grows large for a fixed processor design, the signal propagation time becomes dominant. Only if one ignores the propagation delay, does the classic decimation in frequency algorithm extend optimally to large numbers of samples. If one only considers multiplication and addition time, then an unlimited number of processors, can execute a fast Fourier Transform in O(log n) time. Consideration of propagation delay, shows that communication time grows as $n^{1/2}$ log n and soon dominates. By changing the radix to $n^{1/2}$, and using two stages, execution time is reduced to $O(n^{1/2})$, so that the decimation in frequency algorithm is shown to be nonoptimal.

These results are in keeping with empirical experience. The comparison of various fast Fourier transform algorithms on an assortment of vector concurrent processors[6] demonstrates the advantage of the two-stage algorithm for a large number of samples. As the degree of parallelism grows or where communication is costly, the two-stage algorithm requires less time.

As we continue to improve the speed of our processing, other costs of computation such as communication time become more significant. We must consider these other factors in estimating the time and component cost of high–performance algorithms and in selecting the optimal one for a given situation. With technology typical today, we are seeing these effects in multiprocessor computers that fail to realize the high gross processing speed promised.

---

[1]J.W. Cooley and J.W. Tukey, *Mathematics of Computation*, vol. 19, April 1965, pp. 297-301

[2]Ronald N. Bracewell, **The Fourier Transform and its Applications**, McGraw-Hill, Inc., 1978, p. 370-379

[3]Brian Bakoglu, **Circuit and System Performance on ULSI: Interconnections and Packaging**, Doctoral Thesis, Stanford University, August 14, 1986

[4]Robert W. Keyes, **The Wire-Limited Logic Chip**, *IEEE Journal of Solid–State Circuits*, Vol. SC-17, No. 6, December 1982, p. 1232-1233

[5]Monolithic Memories, Inc., **LSI Databook**, Monolithic Memories, Inc., 1985, p. 10-2

[6]Paul N. Swarztrauber, **Multiprocessor FFTs**, National Center for Atmospheric Research, June 1986